AFRL-RI-RS-TR-2011-219

PHOENIX: SERVICE ORIENTED ARCHITECTURE FOR INFORMATION MANAGEMENT THE "FAWKES" CURSOR-ON-TARGET ROUTER

SEPTEMBER 2011

INTERIM TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2011-219 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/
STEVEN D. FARR
Branch Chief

/s/
JULIE BRICHACEK, Chief
Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
SEP 2011	Interim Technical Report	JAN 2009 – NOV 2010

4. TITLE AND SUBTITLE

PHOENIX: SERVICE ORIENTED ARCHITECTURE FOR INFORMATION MANAGEMENT THE "FAWKES" CURSOR-ON-TARGET ROUTER

5a. CONTRACT NUMBER
In House

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

V. Combs (AFRL), J. Hanna (AFRL), T. Krokowski (RRC), B. Lipa (ITT)

5d. PROJECT NUMBER
S2TS

5e. TASK NUMBER
IH

5f. WORK UNIT NUMBER
03

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

AFRL/RISE, 525 Brooks Road, Rome, NY 13441-4505
ITT, 775 Daedalian Drive, Rome NY 13440
RRC, Ridge Street, Rome NY 13440

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RISE
525 Electronic Parkway
Rome NY 13441

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2011-219

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# 88ABW-2011-0020

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Cursor-on-Target (CoT) is a strategy for enabling DoD systems to communicate much needed time sensitive position or "What, When, Where" (WWW) information. CoT leverages the ubiquitous XML technology and defines a common, terse yet extensible message format for communicating WWW information. A data strategy akin to object oriented decomposition is used to define and manage extensions to the base WWW data. Using this approach CoT can easily and effectively represent BFT, TST, mayday messages, CSAR reports, spot reports, ISR asset tasking, battlefield reservations, and many other tactical battlefield information exchange needs.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			VAUGHN COMBS
U	U	U	UU	26	19b. TELEPHONE NUMBER (*Include area code*) N/A

Table of Contents

Introduction

Cursor-on-Target

"Cursor-on-Target (CoT) is a strategy for enabling DoD systems to communicate much needed time sensitive position or "What, When, Where" (WWW) information. CoT leverages the ubiquitous XML technology and defines a common, terse yet extensible message format for communicating WWW information. A data strategy akin to object oriented decomposition is used to define and manage extensions to the base WWW data. Using this approach CoT can easily and effectively represent BFT, TST, mayday messages, CSAR reports, spot reports, ISR asset tasking, battlefield reservations, and many other tactical battlefield information exchange needs." [4]

CoT messages must adhere to the document standard set forth by the master schema distributed and maintained by MITRE. This section will provide a brief overview of the required fields for a valid CoT message. Where practical, field descriptions were copied "as is" from the CoT schemas in an effort to retain data integrity.

The 'event' node is the parent node of a CoT message and its fields define the "What" and "When" parts of the message. CoT messages are commonly referred to as "events" due to this naming scheme. The required fields of the 'event' node include:

- **uid** - The "uid" attribute is a globally unique name for this specific piece of information. Several "events" may be associated with one UID, but in that case, the latest (ordered by timestamp), overwrites all previous events for that UID.
- **type** - The "type" attribute is a composite of components delimited by the semi-colon character. The first of which are described here:
    - a -Atoms - identifies the fact that this event describes an actual "thing". The "Atoms" portion of the type tree contains CoT defined fields and part of the MIL-STD-2525 type definition. To distinguish MIL-STD-2525 type strings from CoT defined fields, the MIL-STD-2525 types must be represented in all upper case. When using the "Atoms" designation, the next character in the set defines the affiliation of the "thing" as follows:
        - p - Pending
        - u - Unknown
        - a - Assumed friend
        - f - Friend
        - n - Neutral
        - s - Suspect
        - h - Hostile
        - j - Joker
        - k - Faker
        - o - None specified
        - x - Other

    - b - Bits - Events in the "Bit" group carry meta information about raw data sources. For example, range-doppler radar returns or SAR imagery represent classes of information that are "bits".
    - t - Tasking - (requests/orders) Events in this category are generalized requests for service. These may be used to request for data collection, request mensuration of a specific object, order an asset to take action against a specific point.

- **time** - The CoT XML schema includes three time values: time, start, and stale. "time" is a time stamp placed on the event when generated. The format of time, start, and stale are in standard date format (ISO 8601): CCYY-MM-DDThh:mm:ss.ssZ (e.g., 2002-10-05T17:01:14.00Z), where the presence of fractional seconds (including the delimiter) is optional.
- **start** - The "start" attribute defines the starting time of the event's validity interval. The start and stale fields together define an interval in time. It has the same format as time and stale.
- **stale** - The "stale" attribute defines the ending time of the event's validity interval. The start and stale fields together define an interval in time. It has the same format as time and start.</
- **how** - The "how" attribute gives a hint about how the coordinates were generated. It is used specifically to relay a hint about the types of errors that may be expected in the data and to weight the data in systems that fuse multiple inputs.

The fields of the 'point' node describe the "Where" part of the CoT message. Its required fields include:

- **lat** - Latitude based on WGS-84 ellipsoid in signed degree-decimal format (e.g. -33.350000). Range -90 -> +90.
- **lon** - Longitude based on WGS-84 ellipsoid in signed degree-decimal format (e.g. 44.383333). Range -180 -> +180.
- **ce** - Circular Error around point defined by lat and lon fields in meters. Although named ce, this field is intended to define a circular area around the event point, not necessarily an error (e.g. Describing a reservation area is not an "error"). If it is appropriate for the "ce" field to represent an error value (e.g. event describes laser designated target), the value will represent the one sigma point for a zero mean normal (Guassian) distribution. To ignore this field, enter seven nines, 9999999.
- **le** - Linear Error in meters associated with the HAE field. Although named le, this field is intended to define a height range about the event point, not necessarily an error. This field, along with the ce field allow for the definition of a cylindrical volume about the point. If it is appropriate for the "le" field to represent an error (e.g. event describes laser designated target), the value will represent the one sigma point for a zero mean normal (Guassian) distribution. To ignore this field, enter seven nines, 9999999.
- **hae** - HAE acronym for Height above Ellipsoid based on WGS-84 ellipsoid (measured in meters). To ignore this field, enter seven nines, 9999999.

The 'detail' node contains any additional information describing the WWW information described by the CoT message. It may contain any valid XML, but there are several additional child schemas defined for common message extensions. Again, the descriptions of these schemas were copied "as is" from the MITRE schemas to insure data integrity. The provided set of CoT sub-schemas includes:

- *_flow-tags_* - This is a Cursor On Target detail sub-schema that holds "fingerprints" of the system that have processed a particular CoT event. This information aids in the routine of CoT messages along a particular processing chain. Each system that touches a particular CoT event is expected to add its own attribute to this entity. The attribute name should reflect the particular system name, and the value should be the time stamp when the information was sent out from that system. Some illustrative _flow-tags_ attributes are adocs, fbcb2, and tadilj, but the attribute list is not a closed set.
- *contact* - This is a Cursor On Target sub-schema representing communications parameters for contacting a friendly element for human-to-human communications. The objective of this schema is to carry the essential information needed to contact this entity by a variety of means. None of the modes of contact (e.g., e-mail, phone, etc) is required.
- *engage* - This is a Cursor On Target sub-schema for representing weapon/target paring and mission engagement. The objective is to carry the essential information needed to task an asset for strike/csar/recce/etc. The scheme is intentionally not platform specific (e.g., we want to be able to strike a target with air, ground, or naval assets.) "Engage" is a slight misnomre when this sub-schema is applied to ISR or CSAR missions, but the information remains essentially the same.)
- *image* - This is a Cursor On Target sub-schema for abstract image product metadata. It is specifically limited to geographically located (though not necessarily geographically registered) image products. It is not intended to contain all the meta data typically found in the NITF header associated with such images, but rather provides sufficient "hints" about the ISR product to facilitate collection queuing and ipl searching. Full meta data will reside in the NITF header or other IPL-specific schemas. This sub schema borrows from the NITF standard. Note, also, that this subschema presumes is is contained within a CoT base element which provides information about center point, etc. Similarly, the CoT_shape schema can be used to delimit the bounds of the image. Furthermore, this element may contain a base64 encoded image file. In this case, the 'mime' attribute should indicate the image type.
- *link* - This is a Cursor On Target sub-schema for linking to either another CoT event or an arbitrary Internet resource. The objective of this schema is to provide an abstract way to express a relationship between a CoT object and other object. This allows, for example, a sensor point of interest to be linked back to its source, or a PPLI from a wingman to be associated with his flight lead. Linkages are always unidirectional. One entity may have multiple links (i.e., it may be related to multiples other entities). For processing simplicity, it is required that the relationship graphs will directed and acyclic (no cycles). The link, itself, names the relationship (using a hierarchy similar to the CoT type), the UID of the related object (whether CoT or not), possibly provides a URL for retrieving that object.
- *mensuration* - This is a Cursor On Target sub-schema for a mensuration meta-data. This schema encapsulated information of interest of mensuration systems which is not covered in the CoT base schema. This entity is both an "input" and "output" entity. That is, it may be used to provide "hints" to a mensuration system, or it may be used by the mensuration system to record significant information about the mensuration process.
- *remarks* - This is a Cursor On Target sub-schema for a generic remarks (aka "FreeText"). While the use of free text is strongly discouraged (it hampers machine-to-machine communication) it is a pragmatic necessity. This entity attempts to encapsulate "FreeText" in a way that simplifies subsequent machine processing. The content of this entity is presumed to be a human-readable chunk of textual data. The attributes merely aid in the machine handling of the data.

- *request* - This is a Cursor On Target sub-schema for a generic request. This schema contains information common to all requests, specifically where responses should be sent, the overall priority of the request, if immediate wilco/cantco acknowledgment is needed, etc. Detail information for specific request types are carried in sub-schemas nested within this one.
- *sensor* - This is (the root class of) a Cursor On Target sub-schema for a steer-able, staring sensor such as EO, IR, or Radar sensor. The root class is intended to capture only information on the sensor's orientation and field of view is. Details about it's spectrum, sensitivity, resolution, modality, performance, etc., should be captured in a "derived" subschema for that particular type of sensor. All orientation attributes associated with sensor are normalized to a geodesic frame of reference, removing platform factors such as roll, pitch, yaw, etc. Therefore an "azimuth" of 0 means the sensor is pointed north regardless of its platform heading or attitude.
- *shape* - This is a Cursor On Target sub-schema for a generic shape description. Many objects are not adequately represented by the simple "point" object in the CoT base schema. However, it is counterproductive to burden all CoT applications to understand arbitrary shapes, so "shape" is an optional attribute that can be used to communicate between shape-aware appreciations. The "point" object in the base schema must still be populated and the CE and LE fields in the point entity must be set such that the point completely encloses the area described in any shape entity in the detail section. (This is needed so that CoT applications can quickly filter out objects that are clearly outside an area of interest.
- *spatial* - This is a Cursor On Target sub-schema for spatial information of physical entity. It is intended to appear in the detail section of the Cursor On Target schema. It has elements to represent attitude and associated first derivatives (spin). The intention behind the spatial element is to convey the attitude of a body moving through space with respect to its "nominal" flight attitude.
- *status* - This is a Cursor On Target sub-schema designed to contain status information on a friendly entity. It has no attributes of its own, but contains status sub-schemas such as fuel, medical, or weapons status.
    - *fuel* - This is a Cursor On Target sub-schema for representing a platform's fuel status. The objective of this schema is to carry the essential information about a platform's fuel to enable functions such as weapon target pairing, etc. Two types of fuel are represented, burn-able (which the platform can consume) and off-load (which the platform can deliver to another asset.) If multiple types of fuel are carried, the fuel entity may be repeated.
    - *medical* - This is a Cursor On Target sub-schema containing information on the medical status of a friendly object.
    - *weapons* - This is a Cursor On Target sub-schema for representing weapon status information The objective of this schema is to carry the essential information about a weapon to enable functions such as weapon target pairing to be accomplished.
- *track* - This is a Cursor On Target detail sub-schema for track information. The root element and associated attributes of this schema are intended to appear in the detail element of the Cursor On Target schema.
- *uid* - This is a Cursor On Target detail sub-schema that holds the unique ID assigned by each system that processed this event. Most systems (including CoT) have their own method for assigning system-wide unique identifiers for a particular object. In general, it is not possible for a single UID to be used for all systems. This 'uid' entity provides a

common place where each systems can record its particular UID for each CoT event. Like the _flow-tags_ element, each system is responsible for adding its own attribute to this entity. The name of the attribute should represent the system, and the value of the attribute should be the id that system assigned to this CoT object.

Design

This document outlines the specifics of the Fawkes CoT Router Services including technologies utilized and key design decisions. The Router Services consist of three segments: the component packages, the service packages and the Graphical User Interface (GUI) for managing information flowing through the service. The component and service packages map functionally and semantically to their respective Phoenix Architecture packages of the same name and extend many of the basic constructs offered by their respective Phoenix Base Implementation packages of the same name.

Conventions

This document provides both a literal and conceptual design of the Phoenix architecture. The literal architecture is a technical specification defined using UML. The conceptual architecture is a less formal description using plain language and diagrams to provide design concepts and objectives.

Diagram Conventions

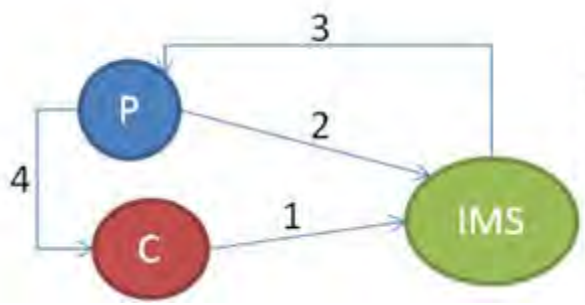
Throughout this document there are a number of non-UML diagrams that are used to illustrate high-level concepts. Samples of these diagrams are shown below along with usage information.

The figure below shows a sample communication between Phoenix entities via channels.

Entity	Meaning	Color
Producer	Produces information.	

Service	Manipulates information.
Consumer	Consumes information.
Actor	A generic term that can mean producer, consumer, or service.
Inquisitor	A type of consumer that queries a service to get information.

The figure below is a sample diagram showing labeled information flow.



1. Consumer registers with IMS.
2. Producer requests consumer list.
3. Producer receives consumer list.
4. Producer delivers data directly to consumers.

Implementation Language

Java was selected as the development language of choice for this project due to several factors:

- Ease of Use and Understanding
- Existing Built-in Features including support for RMI, XML, and Concurrency
- Availability of numerous 3rd party libraries such as Log4J, XPP, among many others
- Development Team Experience

At design time the decision was made to go with the latest version of the Java Software Development Kit (SDK) available, which was Java Developer's Kit (JDK) 6. JDK 7 could not be considered because it is in an early development phase, which introduces too much risk and would inhibit engineering productivity.

Code Conventions and Formatting

Code formatting is a huge issue in a distributed development environment. Formatting has been standardized for the project by applying a standard format configuration file that is enforced through the activation of the Checkstyle plug-in for Eclipse and Maven. The current Eclipse plug-in version is 4.4.2 and the current Maven plug-in version is 2.2. More information about the Checkstyle Eclipse and Maven plug-ins can be found at the Checkstyle web-site: http://checkstyle.sourceforge.net

The current code format is an extension of Sun's suggested standard code conventions for Java applications. Some high level settings of note are a max line length of 120 characters, a max

method size of 150 lines of code, and the application of variable declaration templates consistent with Sun's standards.

FindBugs : Bug Finding and Reporting Plug-in

Discovering bugs in a project is a job assigned to the FindBugs plug-in for Eclipse and Maven. Bugs are reported within Eclipse by the FindBugs views provided by the plug-in while the bugs reported by the Maven plug-in are available for view only through the Maven project module's individual web-sites. The Eclipse FindBugs plug-in can be configured to run during every compilation done by Eclipse while the Maven FindBugs plug-in is only run when Maven builds the module's corresponding web-site. The current versions of the FindBugs plug-in for Eclipse and Maven are 1.3.7 and 1.2, respectively. More information about the FindBugs Eclipse and Maven plug-ins can be found at the FindBugs web-site: http://findbugs.sourceforge.net

PMD

The Maven builder for the project also incorporates the PMD plug-in. This plug-in, run only when Maven builds the corresponding web-site for a project module, checks for possible bugs, dead and suboptimal code, overly complicated conditional expressions, and duplicate code. PMD reports are available via a link on each module's web-site. The current version of PMD Maven plug-in used for this project is 2.4. More information about the PMD plug-in and its capabilities can be found at its web-site: http://pmd.sourceforge.net

Component Package Implementations

An listing of the CoT Router Services component package implementations and their specifics:

- Channel
- Filter
- Expression

Channel

The channel component package for the CoT Router Services implements a bridging mechanism that allow native Phoenix services to receive and transmit standard Cursor-on-Target (CoT) messages using the protocols and transport mechanisms of standard CoT clients. There are three custom channel implementations that support the UDP, TCP, and streaming TCP transport mechanisms most commonly utilized by native CoT clients.

Receiving CoT

The CoT specific channel implementations receive messages via UDP and TCP sockets and convert them to Phoenix information instances. The entire CoT message becomes the metadata for the information instance, the payload is left as null, and the information type name is set to a value read from a configuration file. Default information type name for the CoT Router Service is: "mil.af.rl.cot"

Transmitting CoT

The CoT specific channel implementation transmit messages via UDP and TCP sockets by taking an instance of Phoenix information, stripping out the metadata, and transmitting it. Since the metadata of a Phoenix information instance is set to the complete CoT message (within the CoT Router Services environment), this is all that needs to be done.

Table of Valid CoT-specific Application and Transport Level Protocol Pairs

Application Level Protocol	Transport Level Protocol	Description
tcpcot	tcp	This pairing defines the "streaming" TCP CoT mechanism where a TCP socket is opened and CoT messages are transmitted/received without closing the socket after each write/read.
cot	tcp	This pairing defines the native TCP CoT mechanism where a TCP socket is opened, a single CoT message is transmitted/received, and then the socket is closed.
cot	udp	This pairing defines the native UDP CoT mechanism.

Filter

The filter component package for the CoT Router Services contains the filters implemented for CoT operations. Filters are used to support all of the CoT Router 'edit' operations. These operations edit the CoT messages when they match subscriptions and before they are sent to any of the corresponding consumers. There are also filters defined for decimation and metadata and payload stripping operations.

Implemented Filters

- **Decimation Filter** - This filter will decimate, or provide a skipping function, over the information that is sent through it. For example, when applied to a subscription and with a decimation value set at 2 this filter would only allow every other message to get to the consumers of the subscription. If the value was set to 1 all messages would get to the consumers. And to hammer the point home, if the value was set to 5 every fifth message would be sent to the consumers.
- **Metadata Stripping Filter** - This filter will remove the metadata from the information that is provided to it, but leave the payload and information context alone. Since we currently put the entire CoT message in the metadata and set the payload to null this filter does not have much use as of yet.
- **Monitoring Filter** - This filter allows all CoT messages sent through it to continue untouched, however it will send a copy of each message to a defined CoT output channel that is being used for monitoring purposes.
- **Payload Stripping Filter** - This filter will remove the payload from the information that is provided to it, but leave the metadata and information context alone. Since we

currently put the entire CoT message in the metadata and set the payload to null this filter does not have much use as of yet.

- **Router Operations Filter** - This filter will perform many operations. It always time stamps and UID stamps each CoT message sent through it. In addition to this and based on the CoT edit operations defined for the subscription, this filter may perform several other operations upon the CoT message as well. The edit operations supported by this filter are explained in full below.

CoT Router Edit Operations

delete() – Remove a node from the CoT message.

EXAMPLE:
<edit event.detail._flow-tags_="delete()">

This example, when applied to a subscription, will remove the /event/detail/_flow-tags_ node from every message that matches the subscription's expression and send the modified message to the subscription's recipients.

set() – Set the value of a node to the given value.

EXAMPLE:
<edit point.hae="set(444)">

This example, when applied to a subscription, will set the /event/point/@hae value to be 444 for every message that matches the subscription's expression and send the modified message to the subscription's recipients.

timestamp() – Timestamp the specified node with a date time stamp of the current Zulu time. If the node exists, replace the current value, if not add the node as a new element and set the value to the time stamp.

EXAMPLE:
<edit uid="timestamp()">

rate() – Control the rate of data flow by specifying a rated transmission interval for the subscription's transmitters. Setting this function requires a field be specified in the router GUI, even though the field is irrelevant to the function. This function operates as described below:

etc.
rate(.25) = Send a message every 4 seconds
rate(.5) = Send a message every 2 seconds
rate(1) = Send a message every 1 second
rate(2) = Send two messages every 1 second
etc.

EXAMPLE:
<edit uid="rate(.25)">

dump() – Dump the contents of the message matching the subscription to the router log.

EXAMPLE:
<edit uid="dump()">

watch() – Watch the specified subscription and report the UID of each matching event as they are matched against the subscriptions.

EXAMPLE:
<edit uid="watch()">

Expression

The expression component package for the CoT Router Services defines contexts and expression processors that are specific to CoT information brokering operations.

CoT Router Expression Processors

Cursor-on-Target defines its own expression lanaguage with its own functions and node paths. A complete listing of the CoT Router functions used for expression processing is included below:

Most of the functions listed above are translated directly to XPath. However some were either unable to be tnaslated or deemed to be inappropriate to translate to XPath so a set of custom CoT expression processors were developed to handle these exceptions to the rule. These custom processors are as follow:

- **In File** - A processor that checks if the value at the given path is listed in the identified file. The file is assumed to be a Comma Separated Values (CSV) file. No other type of file is supported per CoT definition of this function.
- **Size** - A processor that checks the size of the CoT message being processed to see if it matches the defined parameters (I.E. is it larger than X).

Category	Tests	Description	Supporting Implementation Class
Test of tag existence	exists	True if field exists in event - 'exists()'	XpathExpressionProcessor
	missing	True if field does not exist in event - 'missing()'	XpathExpressionProcessor
Test if node has certain children	child	True if entity has any children	XpathExpressionProcessor
	hasany	True if entity has any of these children 'detail._flow-tags_=hasany(ncct,adocs,tadilj)'	XpathExpressionProcessor
	hasnone	True if entity has none of these children 'detail._flow-tags_=hasnone(ncct,adocs,tadilj)	XpathExpressionProcessor

		'	
	hasall	True if entity has all of these children 'detail._flow-tags_=hasall(ncct,adocs,tadilj)'	XpathExpressionProcessor
Event expression tests	is	True if event matches any expression - 'is(neutral,friend)'	XpathExpressionProcessor
	isall	True if event matches all expressions - 'isall(neutral,friend)'	XpathExpressionProcessor
	isany	True if event matches any expression - 'isany(neutral,friend)'	XpathExpressionProcessor
	isnot	True if event does not match any expression - 'isnot(neutral,friend)'	XpathExpressionProcessor
Matching against a field	match	True if field does match regular expression - 'match(a-h.*)'	XpathExpressionProcessor
	nomatch	True if field does not match regular expression - 'nomatch(a-h.*)'	XpathExpressionProcessor
Member tests	member	True if field exists in given list of parameters – member(m-g,m-p,m)	XpathExpressionProcessor
Numerical expression tests	n_eq	True if field is numerically equal to value - 'n_eq(10)'	XpathExpressionProcessor
	n_gt	True if field is greater than argument - 'n_gt(27.1)'	XpathExpressionProcessor
	n_in	True if field is in numeric range - 'n_in(100,200)'	XpathExpressionProcessor
	n_lt	True if field is less than argument - 'n_lt(27.1)'	XpathExpressionProcessor
	n_out	True if field is out of numeric range - 'n_out(100,200)'	XpathExpressionProcessor

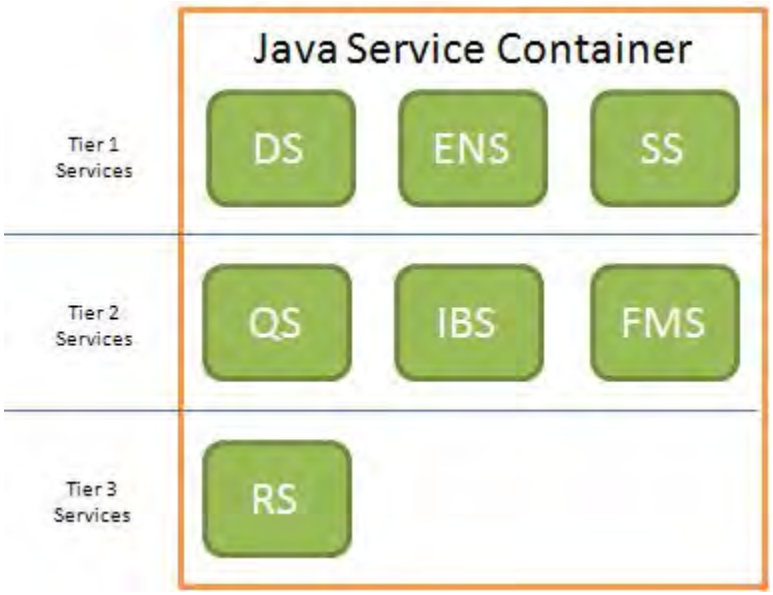
	n_range	True if field is in numeric range - 'n_range(100,200)'	XpathExpressionProcessor
String expression tests	s_eq	True if field is lexicographically equal to argument - 's_eq(dino)'	XpathExpressionProcessor
	s_gt	True if field is lexicographically greater than argument - 's_gt(dino)'	XpathExpressionProcessor
	s_in	True if field is in lexicographic (string) range, inclusive - 's_in(betty,wilma)'	XpathExpressionProcessor
	s_lt	True if field is lexicographically less than argument - 's_lt(dino)'	XpathExpressionProcessor
	s_out	True if field is not in lexicographic (string) range, inclusive - 's_out(betty,wilma)'	XpathExpressionProcessor
	s_range	True if field is in lexicographic (string) range, inclusive - 's_range(betty,wilma)'	XpathExpressionProcessor
Value in file tests	infile	True if field is listed in specified file - 'infile(list.txt)'	InFileExpressionProcessor
	notinfile	True if field is not listed in specified file - 'notinfile(list.txt)'	InFileExpressionProcessor
Test size of XML	larger	True if event's XML representation is larger than this many bytes – larger(1024)	SizeExpressionProcessor
	smaller	True if event's XML representation is smaller than this many bytes – smaller(1024)	SizeExpressionProcessor
Other subscriptions	indirect	True if field matches listed subscriptions' tests – indirect(sub1,sub2)	CotInformationBrokeringServiceWorker
Non-expression tests	links	Search cache for linked events to send to subscription consumers – links()	CotInformationBrokeringServiceWorker

	unblock		Not Implemented

Service Implementations

The Cursor-on-Target (CoT) Router Services (CRS) are comprised of several tiers of Phoenix Base Implementation services that, in some cases, are configured with CoT-specific logic modules that provide functionality not included as part of the Base Implementation. Specifically, the Dissemination, Event Notification, Information Type Management, and Query Services are all standard Base Implementation services while the Information Brokering, Repository, and Submission Services have been configured to use logic modules specific to CoT operations.

The CRS are hosted by the Phoenix Java Service Container (JSC), which provides a simple service execution and management container hosted by a Java Virtual Machine (JVM). This container hosts and maintains the services and their associated connectors.



1 - CRS Services

"The Tier 1 Operational Services directly interact with the edge actors via information and event channels. Control operations upon these services by edge actors are possible, depending upon the security policies being enforced by the implementation. Tier 2 services either do not directly process information or they do not directly communicate with edge actors to either receive or deliver information. Tier 3 services provide basic information management functionalities, but provide interfaces that are not exposed to edge actors. These services are available only through other service proxies and are not intended to provide direct channel support to the edge actors." [5]

Dissemination Service

The CoT Router Services include a Phoenix Base Implementation Dissemination Service (DS). This service is configured to enable event firing in general and output channel status update events in particular. These events are used by the CoT Router GUI to track the status of the output channels to CoT consumers.

Event Notification Service

The Fawkes CoT Router Services include an instance of the Phoenix Base Implementation of the Event Notification Service (ENS). This service supports the CoT Router GUI by delivering status update events for input channels, output channels, and subscriptions.

Filter Management Service

The Filter Management Service (FMS) for the CoT Router Services is the storehouse of filter definitions for the CoT Router. The FMS is used to store and manage the CoT specific filters that have been developed for the CoT Router Services.

Information Brokering Service

The Information Brokering Service (IBS) for the CoT Router Services is an instance of the Phoenix Base Implementation IBS, configured with a set of CoT specific expression processors in addition to the Base Implementation XPath and Expressionless Processors. There are no other differences between the Base Implementation and CoT Router versions of the IBS. The CoT expression processors are described here.

Information Type Management Service

The Information Type Management Service (ITMS) for the CoT Router Services is an instance of the Phoenix Base Implementation of the ITMS. It is used to store the information type definition for the CoT type and to configure the Repository Service at start-up time to store the CoT information type.

The standard information type name chosen for CoT messages is: *mil.af.rl.cot*

The XML Schema Document (XSD) that defines a CoT message's format and syntax is stored by this service for potential validation operations and reference. This schema contains a set of high level attributes and the definition for both the root level node of a CoT message and the child node that defines the spatial aspects of the message. For more details about the contents of a CoT message as defined by the XSD, see the Introduction.

Query Service

The Query Service (QS) for the CoT Router Services is an instance of the Phoenix Base Implementation QS. It is used by the Submission Service (SS) when a CoT query message is submitted. The QS receives the execute query control invocation and farms it out the underlying Repository Service (RS). It is possible to configure more than one RS for a CoT Services deployment, though most currently utilize on a single RS. All CoT queries are assumed to be asynchronous in execution. This was done to allow the user who issued the query to do other things while awaiting their result set.

Repository Service

The Repository Service (RS) for the CoT Router Services is an instance of the Phoenix Base Implementation RS that is configured with a CoT specific implementation of a PostGIS Repository.

PostGIS Repository

PostGIS provides a set of custom data types and stored procedures that help users efficiently manage geospatial data. Without PostGIS, users would be required to formulate complex queries that would be subject to user error and largely inefficient when compared to the feature set provided by a spatial extension. In order for PostGIS to function, a valid install of the PostgreSQL RDBMS must exist on the target system. The repository service interfaces with PostgreSQL and PostGIS through a standard Java Database Connector (JDBC) API.

Storing Information

The underlying PostgreSQL RDBMS ensures that each database transaction is atomic in nature. As a result of the atomicity it is implied that multiple clients can store and query information simultaneously without having to provision for concurrent operations. Information stored using the PostGIS repository assumes the following table layout / column definitions outlined below:

Column Definitions

The scripts needed to create the database table needed by PostGIS repository service are included as a series of resources with the repository service (in the directory repository/src/main/resources).

- id - A unique identifier. Populated automatically by the PostgreSQL RDBMS
- uid - The UID for a CoT message
- cot_type - The type of CoT message (e.g., t-x-i)
- start - A timestamp containing the start time for a CoT message
- time - A timestamp containing the defined time (e.g., the time the message was created) for a CoT message
- stale - A timestamp containing the stale time for a CoT message
- serialized_obj - The serialized Java object representing the full CoT message
- event_pt - The geographical point defined in the CoT message

To enhance query performance, indices have been created on the uid, cot_type, time, and event_pt columns. The event_pt column and its associated index rely on a standard WGS-84 ellipsoid and require all queries / geospatial functions to rely on the same ellipsoid.

Querying Information

Clients of the PostGIS Repository Service must use the InformationQueryContextInterface to construct a query object. The predicate supplied to the InformationQueryContextInterface must be a valid Cursor-on-Target (CoT) message.

Sample CoT Query Message

```
<detail>
  <request notify="127.0.0.1:9999:udp">
    <tests>
      <test uid="COT-UID-1" start="range(DTTM-1, DTTM-2)" />
      <test uid="COT-UID-2" />
      </tests>
      </request>
      <shape>
      <polyline>
        <vertex lat="10.1" lon="20.2" />
        <vertex lat="10.2" lon="20.3" />
        <vertex lat="10.3" lon="20.4" />
        <vertex lat="10.4" lon="20.5" />
        </polyline>
      </shape>
</detail>
```

The above sample represents a portion of a CoT message that will query the PostGIS repository service for information within a rectangular area of interest. In addition to storing and querying geospatial information, the PostGIS repository service can query traditional CoT attributes (see table schema above) such as UID, start time, and type. The `<tests>` element can contain one or more `<test>` elements that define an attribute driven test. In the sample above, the two test elements define a query that targets information with that has uid of COT-UID-1 and a start date range or information that has a uid of COT-UID-2.

Types of Geospatial Queries

Clients can perform spatial queries that are based off of lines, rectangles, and circles. The following section outlines the XML the client must provide inside the <shape> element for each type of geospatial query previously mentioned:

- Rectangle - A set of 4 `<vertex>` elements that define 4 points representing a rectangular area of interest. The `<polyline>` element must contain the "closed" attribute and its value must be "true".
- Line - A series of 2 or more `<vertex>` elements inside of a `<polyline>` element.
- Circle - Replace the `<polyline>` element with the `<circle>` element.

Deleting Information

Aside from deleting information based on type alone (via the `destroyCollectionForInformationType()`), clients can delete information from PostGIS by providing an instance of the InformationQueryContextInterface with a SQL expression representing the data to be removed (e.g., `DELETE * FROM cot_router WHERE UID='AIMS.1'`).

Known Limitations

The database table mentioned above would need to be modified if additional CoT attributes needed to be exposed or hidden. Since PostGIS relies on a RDBMS, all of the data must be homogenous.

Future Work

Support for the ability to query polygons would be trivial to implement and may provide significant value to the existing repository service. Since PostGIS conforms to the Open GIS Consortium standards for representing "simple features" in a database, an interface to the repository service could be created so that analysis / visualization tools could be plugged directly into the repository service instead of having to setup a client that uses CoT as a message protocol.

Submission Service

The Submission Service (SS) for the CoT Router Services is an instance of the Phoenix Base Implementation SS configured with a CoT Router specific Submission Timer Based Buffer. As everyone knows, the 'doSpend' method of the Timer Based Buffer is currently where the service-specific logic resides for each service.

CoT Submission Timer Based Buffer

The CoT specific buffer for information submission must inspect each submitted information instance to determine whether it is a control traffic message or an information message. Control traffic messages are multiplexed out of the information message stream by type matching. The CoT messages all contain the requried attribute field named 'type' which has a large set of valid entries. Each individual control traffic message has a specific type assigned to it. Supported control traffic messages and their CoT types are:

Message Description	CoT Type(s)
Create Subscription	't-b' or 't-b-a'
Destroy Subscription	't-b-c'
Execute Query	't-x-i'
Cancel Query Execution	't-u-z'
Update Query Result Set Transmission Rate	't-x-q-t'

Requirements

The Fawkes CoT Router Services shall have the following requirements:

1. Support all Cursor-on-Target Router operations for message brokering and editing.
2. Support all Cursor-on-Target Router administrative operations for monitoring and control purposes.
3. Support the storage and retrieval of Cursor-on-Target messages.
4. Provide mechanisms for service survival in the event of a sudden catastrophic computing event.
5. Provide mechanisms for information survival in the event of a sudden catastrophic computing event.
6. Provide mechanisms for information survival in the event of a sudden catastrophic environmental event.

Testing

The Fawkes Cursor-on-Target Router Services shall be thoroughly unit, integration, and performance tested using a variety of commercially available and home-grown testing products and methodologies.

Unit Testing

Unit testing of all developed project components and services will be accomplished through the use of the JUnit testing framework. JUnit was chosen because it is becoming the de facto standard within the Java development world for unit testing and it has well-supported plug-ins for both the Eclipse IDE and the Maven build environment. The current version of JUnit used for unit level testing is 4.4.

Unit test coverage reports will be generated by the Cobertura test analysis tool. This tool plugs into the Maven build environment and will be configured to run whenever the web-site is built for a Maven project module. Cobertura offers insight as to what executable lines of code have been tested, how many times they have been executed, and which conditions of a conditional branch have been satisfied by the tests being run. The reports generated divide executable code coverage numbers and conditional branch coverage numbers. The current version of Cobertura Maven plug-in being used is 2.2.

A sample Cobertura report is shown below.

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	47	83%	969/1168	87%	218/250	1.989
mil.af.rl.phoenix.channel	7	99%	160/162	100%	36/36	1.029
mil.af.rl.phoenix.channel.control.connectors	1	100%	2/2	N/A	N/A	1
mil.af.rl.phoenix.channel.control.stubs	1	100%	2/2	N/A	N/A	1
mil.af.rl.phoenix.channel.data	5	96%	72/75	83%	15/18	1
mil.af.rl.phoenix.channel.data.information	3	97%	38/39	100%	10/10	0
mil.af.rl.phoenix.channel.data.information.block	2	100%	25/25	100%	8/8	2
mil.af.rl.phoenix.channel.data.information.mocket	3	88%	59/67	100%	16/16	3
mil.af.rl.phoenix.channel.data.information.stream	2	100%	25/25	100%	8/8	2
mil.af.rl.phoenix.channel.data.transport	6	78%	66/85	88%	7/8	1
mil.af.rl.phoenix.channel.data.transport.mockets	5	62%	156/250	71%	30/42	5.714
mil.af.rl.phoenix.channel.data.transport.tcp	3	83%	79/94	83%	20/24	3.308
mil.af.rl.phoenix.channel.data.transport.udp	2	79%	78/99	100%	20/20	0
mil.af.rl.phoenix.channel.service	3	90%	114/126	70%	21/30	0
mil.af.rl.phoenix.channel.util	4	80%	44/111	90%	27/30	3.667

Report generated by Cobertura 1.9 on 4/21/09 8:00 AM.

Integration Testing

Integration testing of the CoT Router Services will be done using JUnit tests within their own Maven module within the Router Services project. There will be two tests that are identical in functionality but different in set up. One test will set up all services within the test JVM based on hard-coded configurations and pairings. The other test will set up all services by reading and applying a Spring configuration file processed by the Base Implementation Java Services Container (JSC).

Functionally both tests will set up the required services, register a subscription for CoT information, submit a CoT message to the Submission Service and wait for it to be delivered to a test consumer (also set up by the tests). This will validate the functionality and integrity of the CoT Router Services when deployed as a full suite. Both tests will also setup a Repository Service, but no query will be tested here since the control traffic for CoT queries is both complex and due to change in the (hopefully) near future.

Further testing of functionality can be accomplished by running the JUnit test labeled as "For External Clients" and found within the unit tests for the CoT Router Services GUI. Testing the Router Services via this avenue requires the latest installation of the CoT AIMS FalconView plug-in (which obviously necessitates the installation of FalconView).

Reference

Documents

1. Lipa, B. "Berkeley Overview". *AFRL In-House Research (Non-Published)*, May, 2009.
2. Sun MicroSystems, Inc. "Java Code Conventions". *http://java.sun.com/docs/codeconv/CodeConventions.pdf*, 12 September 1997.
3. Oracle. "2.5 Release Overview". *http://www.oracle.com/technology/documentation/berkeley-db/xml/ref_xml/changelog/2.5.html*, September, 2009.
4. MITRE. "CoT Wiki". *http://cot.mitre.org/bin/view/CoT*, 2010.
5. Combs, V. Hanna, J. Lipa, B. Pape, S. Reilly, J. "Phoenix Base Implementation Technical Report". *AFRL In-House Research (Non-Published)*, January, 2010.

Terms and Acronyms

The table below gives a brief description of important terms and acronyms used in this document. For definition of Phoenix Architecture terms and acronyms refer to its specification.

Term/Acronym	Meaning
AFRL	Air Force Research Laboratory.
API	Application Programming Interface.
CoT	Cursor-on-Target.
DLL	Dynamic Link Library.
DoD	Department of Defense.
DS	Dissemination Service.
ENS	Event Notification Service.
FMS	Filter Management Service.
GUI	Graphical User Interface.
HTTP	Hypertext Transfer Protocol.
IBS	Information Brokering Service.
IM	Information Management.
ITMS	Information Type Management Service.
JAR	Java Archive.
JSC	Java Service Container.
JVM	Java Virtual Machine.
NXD	Native XML Database.
PMD	No official definition, several unofficial including *Programming Mistake Detector*.

QS	Query Service.
RMI	Remote Method Invocation.
RS	Repository Service.
SOA	Service Oriented Architecture.
SS	Submission Service.
TCP	Transport Control Protocol.
UDP	User Datagram Protocol.
UI	User Interface.
WWW	What, When, Where.
XML	Extensible Markup Language.
XPP	XML Pull Parser.
XSD	XML Schema Document.
XSLT	Extensible Stylesheet Language Transformations.

Releases

This section lists all releases made of the Fawkes Cursor-on-Target Router Services including short descriptions of why each release was created.

Version	Release Description
0.0.1	Internal release used to setup and configure the release process.
0.0.2	Internal release used to trouble shoot the release process.
0.0.3	First full release of the CoT Router Services for external use in the MARTI projects. This release does not include the Router GUI or the "hardened" Berkeley Repository.
1.0.0	First release of the Router with the bundled GUI and the "hardened" Berkeley Repository.

1.0.1	Release of router made after operational testing. First release of the PostgreSQL and PostGIS repository which replaced Berkeley.
1.0.2	Second release of PostgreSQL and PostGIS repository implementation. Now supports Route Scout queries via a CoT XML sub-schema that AFRL defined.
1.0.3	Implemented the indirect() CoT Router function, the Image Chipper, an enhanced PostGIS database (for Route Scout queries), and upgraded channels and other mechanisms to match the 1.2.1 version of the Phoenix Base Implementation.
1.0.4	Released in synchrony with the Base Implementation version 1.2.2.
1.0.5	Released in synchrony with the Base Implementation version 1.2.3.
1.0.6	Version number skipped to synchronize release versions of XPP CoT and Router projects.
1.0.7	Released in synchrony with the Base Implementation version 1.2.4.